

**Esercizio 1**

Sintetizzare in forma NOR ottima le funzioni:

$$f_1 = \Sigma (0,2,6,8,10,15)$$
$$f_2 = \Sigma (0,1,5,6,9,11,14,15)$$

**Esercizio 2**

Una rete sequenziale analizza vettori binari di 5 bit e se rileva in ciascun vettore un solo bit ad 1 e gli altri quattro a 0 produce 1 in uscita e lo mantiene per due periodi di clock consecutivi; altrimenti l'uscita rimane 0. Progettare la rete facendo uso di flip-flop JK.

**Esercizio 3**

Progettare un circuito digitale che, dati tre interi A, B, C di  $n$  bit, con  $A < B$ , esegue i seguenti calcoli:

- se  $C < A$ , l'uscita U assume il valore di A;
- se  $A \leq C \leq \lceil (A+B)/2 \rceil$ ,  $U = \lceil (A+B)/2 \rceil$ ;
- se  $\lceil (A+B)/2 \rceil < C \leq B$ ,  $U = B$ ;
- se  $C > B$ ,  $U = C \bmod B$

I precedenti esercizi valgono 50/100 del punteggio totale della prova scritta di Architettura dei Calcolatori 1.

I seguenti esercizi valgono 50/100 della prova scritta di Architettura dei Calcolatori 1, ovvero i 50/50 della prova scritta di Calcolatori Elettronici 1 ovvero Calcolatori Elettronici TLC/GES.

4) [18] Si scriva in assembly (MIPS o x86) il codice della routine di gestione dell'interrupt e della routine di inizializzazione della porta seriale per la gestione ad interrupt generato in seguito all'arrivo di dati a 8 bit (parita' di "uni" pari, 1 bit di stop) sulla porta seriale 16550A. Il carattere deve essere letto supponendo che la porta seriale sia mappata in memoria all'indirizzo 0x02F8 e il dato letto deve essere trasferito in un buffer che si trova a partire dall'indirizzo 0x8002'0000. Tale buffer puo' contenere al massimo 16 bytes.

5) [12] Si consideri una cache di dimensione 512B e a 2 vie di tipo write-back. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 3423, 7356, 4557, 1290, 7364, 1789, 1893, 1088, 1019, 1290, 1227, 2902, 2903, 2890, 2160, 3185, 3187, 3101, 3107, 3108, 3912. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento.

6) [20] (v. pagina successiva)

6) [20] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), **rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** (riportate in calce, per riferimento). In alternativa, si usi l'assembly x86 anzichè MIPS. Le funzioni non definite sono da considerare funzioni esterne al programma.

```
typedef struct {
    int idx;
    float val;
    char name[32];
} element;

element container[1000];
int hist[100];

void histogram(element *A, int *bin, int length){
    int i=0;
    for (i = 0; i < length; i++){
        if (A[i].val >= 3.1) bin[A[i].idx]++;
        else strcpy(A[i].name, "Bad element");
    }
}

main () {
    int k=0, n;
    read_int(&n);
    histogram(container, hist, n);
    while (k < n) {
        print_str(container[k].name);
        print_float(container[k].val);
        ++k;
    }
}
```

**MIPS instructions**

Instruction	Example	Meaning	Comments
add	add \$1,\$2,\$3	\$1 = \$2 + \$3	3 operands; exception possible
subtract	sub \$1,\$2,\$3	\$1 = \$2 - \$3	3 operands; exception possible
add immediate	addi \$1,\$2,100	\$1 = \$2 + 100	+ constant; exception possible
subtract immediate	subi \$1,\$2,100	\$1 = \$2 - 100	- constant; exception possible
multiplication	mult \$1, \$2	(HI,LO)= \$1 x \$2	64-bit Signed Product ; result in HI, LO
division	div \$1, \$2	HI= \$1 % \$2, LO = \$1 / \$2	Signed division
move from Hi	mfhi \$1	\$1 = HI	Create copy of HI
move from Lo	mflo \$1	\$1 = LO	Create copy of LO
and	and \$1,\$2,\$3	\$1 = \$2 & \$3	3 register operands; Logical AND
or	or \$1,\$2,\$3	\$1 = \$2   \$3	3 register operands; Logical OR
nor	nor \$1,\$2,\$3	\$1 = ~( \$2   \$3)	3 register operands; Logical NOR
xor	xor \$1,\$2,\$3	\$1 = \$2 ^ \$3	3 register operands; Logical XOR
and immediate	andi \$1,\$2,100	\$1 = \$2 & 100	Logical AND register, constant
or immediate	ori \$1,\$2,100	\$1 = \$2   100	Logical OR register, constant
xor immediate	xori \$1,\$2,100	\$1 = \$2 ^ 100	Logical XOR register, constant
shift left logical	sll \$1,\$2,10	\$1 = \$2 << 10	Shift left by constant
shift right logical	srl \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant
load word	lw \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
load byte	lb \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to register
load byte unsigned	lbu \$1,100(\$2)	\$1 = Memory[\$2+100]	Data from memory to reg.; no sign extension
store word	sw \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
store byte	sb \$1,100(\$2)	Memory[\$2+100] = \$1	Data from register to memory
load address	la \$1,var	\$1 = &var	Load variable address
branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100	Equal test; PC relative branch
branch on not equal	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100	Not equal test; PC relative
set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; 2's complement
set on less than immediate	slti \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare < constant; 2's complement
set on less than unsigned	sltu \$1,\$2,\$3	if (\$2 < \$3) \$1 = 1; else \$1 = 0	Compare less than; natural number
set on less than imm. unsigned	sltiu \$1,\$2,100	if (\$2 < 100) \$1 = 1; else \$1 = 0	Compare constant; natural number
jump	j 10000	go to 10000	Jump to target address
jump register	jr \$31	go to \$31	For switch, procedure return
jump and link	jal 10000	\$31 = PC + 4; go to 10000	For procedure call

**Register Usage**

Name	Register Num.	Usage
\$zero	0	The constant value 0
\$\$0-\$\$7	16-23	Saved
\$\$0-\$\$9	8-15,24-25	Temporaires
\$\$0-\$\$3	4-7	Arguments

Name	Register Num.	Usage
\$\$0-\$\$1	2-3	Results
\$\$fp, \$\$sp	30,29	frame pointer, stack pointer
\$\$ra, \$\$gp	31,28	return address, global pointer
\$\$k0-\$\$k1	26,27	Kernel usage

Name	Usage
\$\$f0, \$\$f1, ..., \$\$f31	Single precision floating point registers
\$\$f0, \$\$f2, ..., \$\$f30	Double precision floating point registers

**System calls**

Service Name	Service Num. (\$v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	\$a0=integer to print	---
print_float	2	\$f12=float to print	---
print_string	4	\$a0=address of ASCII string to print	---
read_int	5	---	\$\$v0=read integer
sbrk	9	\$a0=Number of bytes to be allocated	\$\$v0=pointer to the allocated memory
exit	10	---	---